Information Retrieval
Assignment-II

Rishabh Deshmukh(13349)

February 2017

1 Introduction

This is the report for the second assignment of this course in which I have designed a search engine(with basic functionalities) similar to yippy.com using **Lucene 6.3.0** and **Mallet** library. For ranking, I have used a model which is a function of **tf-idf** score and **probabilistic** score. Both collectively defines the final score of a document-query pair.

1.1 Brief Introduction of yippy search engine

yippy is a search engine which provides clusters of results in addtion to the unclustered results. It uses probabilistic score in addition to the tf-idf score to form clusters of the relevant documents. This type of structure would be useful in situations where users submits an ill-posed query.

1.2 Brief Introduction of Mallet library

Mallet is a library for Java, which provides various functionalities for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text. In this assignment, I have used **Mallet** for topic modeling. For **Topic Modeling** it contains efficient, sampling-based implementations of Latent Dirichlet Allocation, Pachinko Allocation, and Hierarchical LDA.

2 Work Accomplished

2.1 Topic Modeling

Topic model is a model which extracts the underlying **contexts** that occur in a corpus of documents. The process involved in creating topic model is called topic modeling. Typically, we can say that a document consists of various topics in different proportions. A topic can be considered as a collection of similar words.
In this assignment, I used **Mallet** library for topic modeling which gives me some files like topic-keys, document-topic-probability etc. These files are required to compute probabilistic score of a document-query term.

2.2 Ranking

From the work in Assignment-I, we can easily get **tf-idf** by using **score** attribute of **ScoreDoc** class. Now in addition to tf-idf score we also need not compute the probabilistic score, which can be computed as follows :

$$p(Doc|query) = \sum_{context} p(Doc|query)$$
$$p(Doc|query) = \sum_{context} p(Doc|context)p(context|query)$$

If query contains several terms then probabilistic score will be defined as follows:

$$p(Doc|query) = \prod_{term} p(Doc|term)$$
$$p(Doc|query) = \prod_{term} \sum_{context} p(Doc, context|term)$$
$$p(Doc|query) = \prod_{term} \sum_{contexts} p(Doc|context) * p(context|term)$$

The rank of a document is determined by some function of tf-idf score and probabilistic score.

$$finale_{score} = f(Tf - Idf_{score}, Prob_{score})$$

3 Observations

Following are the functions and queries which I have used for comparison:
Query1: central reserve police force.
Query2: action movies.

3.1 Ranking Schemes

3.1.1 Product

$$final_{score} = tfidf * prob_{score}$$

Query1
Highest Ranked Document Title: Central forces to conduct search operations for illegal arms.
Lowest Ranked Document Title: Ponnam's resignation sought.
Query2
Highest Ranked Document Title: Action, romance and more.
Lowest Ranked Document Title: A Bruin-ous tragedy.

3.1.2 Sum

$$final_{score} = tfidf + prob_{score}$$

Query1
Highest Ranked Document Title: Security forces leave for West Bengal.
Lowest Ranked Document Title: IAF invites applications for education instructor.
Query2
Highest Ranked Document Title: Channels.
Lowest Ranked Document Title: On a new high.

3.1.3 Convex Combination

$$final_{score} = a * tfidf + (a - 1) * prob_{score}$$

$$0 < a < 1$$

Query1
Highest Ranked Document Title: Security forces leave for West Bengal.
Lowest Ranked Document Title: IAF invites applications for education

instructor.

Query2
Highest Ranked Document Title: Channels.
Lowest Ranked Document Title: On a new high.

3.2 Conclusion

In my implementation I am getting same set of results for functions **sum** and **Convex Combination** when value of **a** is nearly equal to 0.5. But **Product** function produces different set of results. For Query 1 and 2, **Product** function gives better(more relevant) results as compared to other functions. I came to this conclusion by observing the highest and lowest rank documents retrieved using different ranking scheme to compute final score.